Can be stored in a 2708 EPROM:

2650 utility programs

Here are five utility programs for the 2650 microcomputer, suitable for loading into a 2708 EPROM using the programmer recently described. The routines allow you to perform hex listings, enter programs rapidly in hex from the keyboard, search memory blocks for an instruction, move program or data blocks in memory, and verify program tapes. A number of useful subroutines are also available for use by other programs.

by DAVID EDWARDS

The routines presented in this article are modifications of those originally presented on the Philips/Electronics Australia software record, described in the April 1978 issue. The original routines are quite useful, but have one disadvantage: they have to be loaded into memory every time that the computer is switched on.

By having them stored permanently in a ROM, however, you can avoid this trouble, and make them available for use at a moment's notice. So after completing the EPROM Programmer, my thoughts immediately turned towards these routines, and whether they could be stored in a ROM.

My first idea was to have the EPROM occupy the uppermost memory locations, i.e. from X'7C00 to X'7FFF, so as to leave all of the space below this for memory expansion. However, when I examined the programs in greater detail, I realised that it was necessary to have a small amount of RAM available in the same page as the EPROM, because of the limitations in 2650 memory reference instructions.

The additional hardware required to shift 1k of the existing RAM up into page 3 proved to be too complicated, so I compromised, and decided to put the EPROM at locations X'3C00 to X'3FFF inclusive — i.e., at the top of page 1. The modifications to achieve RAM in this page then became quite simple, involving only one extra gate.

My system at the moment has page 0 completely filled, with the 1K PIPBUG ROM at the bottom of the page, and 7K of RAM filling up the remainder. This RAM is mounted on the prototype 8K RAM board (see December 1978), with pairs of 2114s occupying all locations except those corresponding to the addresses occupied by PIPBUG.

Note that this involves a rearrangement of the high-order address decoding. The 74LS138 decoder on the expansion board is used as the page decoder, and controls the 74LS138s on both the RAM board and the CPU board. The 74LS138 on the RAM board becomes the page 0 decoder, while that on the CPU board becomes the page 1 decoder. Refer to Fig. 1 for a diagram of the wiring.

The chip select signal for PIPBUG is now obtained from the 74LS138 on the RAM board, while the four "spare" RAM pairs on the CPU board are controlled by the 74LS138 on that board. Strictly speaking, only three of these pairs should be used, to avoid overloading the address bus, but in practice we have found that all four pairs can be used without problems.

It is now necessary to disable the main data buffers whenever either PIPBUG or the four RAM pairs are selected. This is the function of the additional gate, the 74LS30 shown in Fig. 1. This is an eight input gate, and is used to replace the inverter provided on the expansion board. It can be mounted on a small piece of Veroboard.

These modifications allow a maximum of 13K of memory to be used, including 11K of RAM. PIPBUG occupies locations X'0000 to X'03FF, RAM from X'0400 to X'2FFF, and the EPROM from

```
Ø6ØØ CD ØF FA CE ØF FB 17 76 4Ø 77 Ø2 75 18 3F Ø2 DB
Ø61Ø 3B 6E 3B FA 3B ØF CD ØF FC CE ØF FD 3B F8 CD ØF
Ø62Ø FE CE ØF FF 17 DA Ø2 D9 ØØ 17 ØD ØF FA ØE ØF FB
Ø63Ø 3B 73 3B 4C ED ØF FC 16 EE ØF FD 17 3F ØØ 8A ØD
Ø64Ø ØF FA 3F Ø2 69 ØD ØF FB 3B F9 Ø4 2Ø 3F Ø2 B4 17
Ø65Ø 3F 5C Ø7 3B 67 ØD 8F FA 3B E9 3B 6E 3B 4C 9E ØØ
Ø66Ø 22 ØC ØF FB 4A [illegible] 1B 69 3B E5 3B CF ØC 3F
Ø67Ø FA EC ØF FE 98 [illegible] ØF EF FA EC ØF F8 98 Ø3
Ø68Ø 3F 3C 3C 3F 3C 2A 9A D7 1B 64 3B C5 3B F3 3F Ø2
Ø69Ø 86 C3 3F Ø2 B4 E7 Ø7 18 C6 E7 2Ø 18 ØB ØC ØF FF
Ø6AØ CC ØF FE CF ØF FF 1B 66 ØC ØF FE 3F Ø2 46 D3 D3
Ø6BØ D3 D3 CF ØF FE ØC ØF FF 3B F2 6F ØF FE CF 8F FA
Ø6CØ 3B C2 ØC ØF FB 44 ØF 18 43 1B 43 3F ØØ 8A 3F 3C
Ø6DØ ØØ ØC 3F FA 14 3F Ø2 B4 3F 3C 2A 1B 71 76 4Ø 3F
Ø6EØ Ø2 86 E4 3A 98 79 2Ø C8 97 3F Ø2 24 CD ØF FA 3B
Ø6FØ F9 CD ØF FB 3B F4 59 ØE Ø5 3D Ø6 31 3B 95 9B 22
Ø7ØØ Ø4 2C Ø4 28 Ø4 29 C9 FA 3B EØ Ø8 F4 18 Ø9 Ø5 3D
Ø71Ø Ø6 34 3F 3C CB 9B 22 C3 CE EA 3B CE ØB E6 EB E2
Ø72Ø 18 Ø8 Ø1 EF EF FA 98 66 DB 6E Ø8 D4 98 6Ø 1F 3C
Ø73Ø DF 4F 4B ØØ 46 41 55 4C 54 59 ØØ 3F 3C Ø7 ED ØF
Ø74Ø FA 19 84 EE ØF FB 1D 3D 84 ØC 8F FA CC 8F FE 3F
Ø75Ø 3C 2A 9E ØØ 22 3B Ø7 3F 3C 25 3B Ø9 1B 6B ØD ØF
Ø76Ø FE ØE ØF FF 17 CD ØF FE CE ØF FF 17 ØD ØF FC ØE
Ø77Ø ØF FD 3B Ø7 CD ØF FC CE ØF FD 17 FA ØØ E6 FF 98
Ø78Ø Ø2 F9 ØØ 17 3B 66 77 Ø9 3B 54 AE ØF FB AD ØF FA
Ø79Ø 75 Ø1 8E ØF FD 8D ØF FC 3E 4B 75 Ø8 ØC 8F FC CC
Ø7AØ 8F FE 3F 3D 5E 3B 54 3F 3D 65 3B 4Ø ØC 8F FC CC
Ø7BØ 3F FE ED ØF FA 19 6B EE ØF FB 19 66 9B 22
```

FIG. 2

When in ROM, the programs must reside at location X'3C00 to X'3DBD.

X'3C00 to X'3FFF. This should allow quite large programs to be run.

The uppermost RAM locations can be reserved for scratchpad use by programs in the ROM. Only six locations are required by the programs presented in this article, so this leaves nearly 11K of RAM available for your programs.

Now that the hardware has been sorted out, we can discuss the programs themselves. These use PIPBUG routines GNUM, CRLF, BOUT, COUT, LKUP, CHIN and BIN, as well as RAM locations CNT, BCC and MCNT.

The first program provided is titled HEX LIST. This produces a hexadecimal listing of any desired memory block, with each line consisting of an address followed by 16 data bytes. To call this routine, type G3C50 AAAA BBBB cr, where A is the start address of the memory area to be dumped, and B is the end address. The listing will include the specified start and end addresses.

If you wish to have fewer data bytes per line, change the contents of location X'3C65 to the appropriate hexadecimal number before you burn the EPROM.

The second routine is called SEARCH. It will list all locations within a given memory block that match a given test pair of data bytes. The matching addresses are printed out in a single column. To call this program, type G3C6A AAAA BBBB XXYY cr, where A and B are the start and end addresses of the range to be searched, and XXYY is the test pattern.

The addresses printed out are those of the first byte of the matched pairs. The search is inclusive, and includes the start and end addresses.

The third program is called HEXIN, and will enable data or programs to be entered into RAM much faster than using the PIPBUG "A" routine. To call the program, type G3C8A AAAA cr, where A is the address of the first RAM location at which bytes are to be entered.

The program will respond by printing out the start address on a new line, and then wait for you to enter hexadecimal characters. Bytes are separated by spaces, and only the last two characters entered before a space are accepted by the program. This means that if you make a mistake, you can simply type in the correct characters before typing the space.

After 16 bytes have been accepted, the program will give a CRLF, and then print the current address at the start of the new line. In this way, if you are careful, you will produce a hex listing as you input the bytes. To terminate the entry mode, type a control-G "BELL" after the space entering the last byte.

The fourth program is titled VERIFY, and is used to check that a PIPBUG absolute object format dump tape is correct and contains no errors, before the master in RAM is destroyed. To use the program, simply type G3CDD cr, and

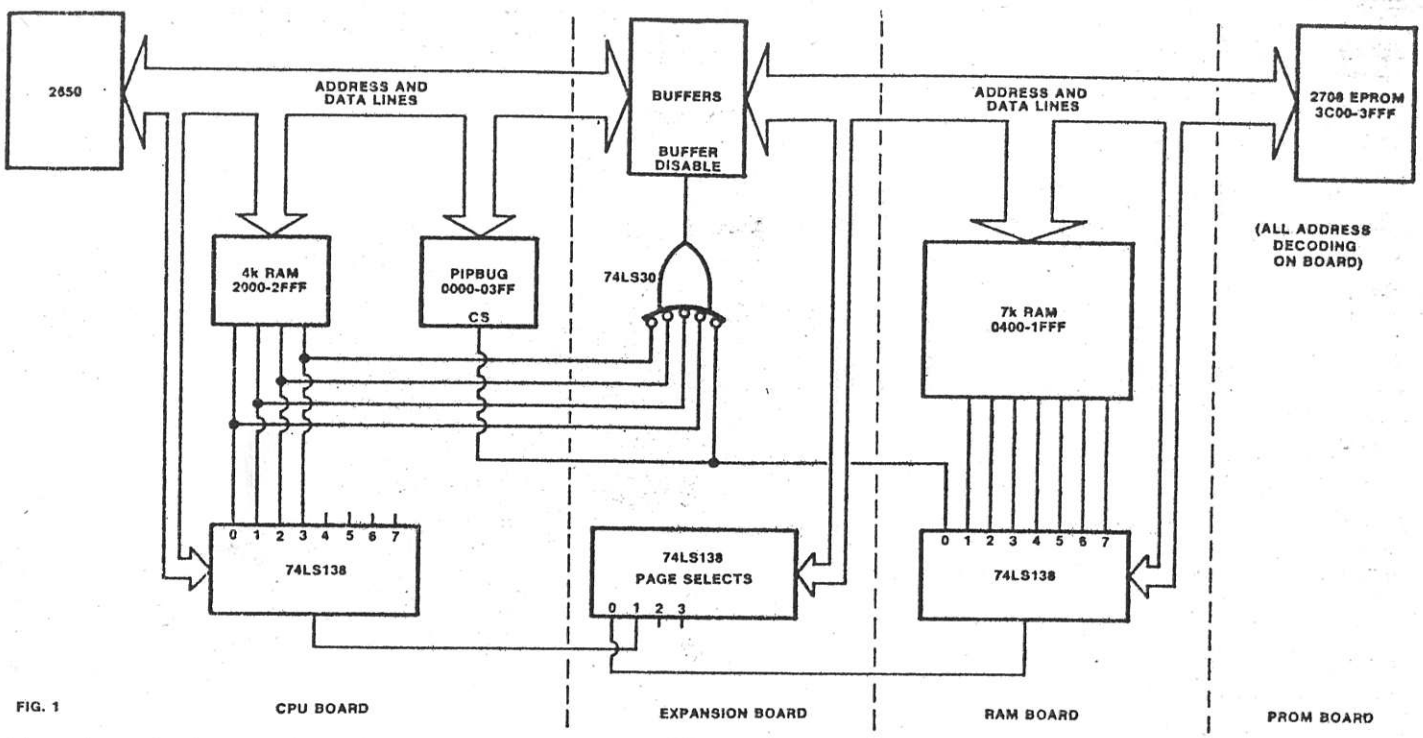

FIG. 1

The schematic diagram shows how the author's system is configured.

then play back the tape to be checked. The program will then read from the tape, and compare its contents with those of the appropriate section of RAM.

If all is correct, the program will respond with the message "OK". If a fault is found, the message "FAULTY" will be printed. This program can only be used to check 110 baud tapes produced by the PIPBUG Dump command. The RAM dumped must still be in memory when the verification is performed, of course.

The fifth and final program is called MOVE. It will shift a specified block of memory to any other location in memory. A memory block can be any size, and can be moved either upwards or downwards in memory by any amount. To use the program, type G3D3B AAAA BBBB CCCC cr.

A and B represent the start and finish locations of the block of memory to be moved, and C represents the new start location. The program will move the memory starting at A and ending at B so that it starts at C and ends at C + A — B. The original memory block will only be changed if the new locations overlap the old locations.

The MOVE program can be used to copy memory from one page to another page, and can also move blocks straddling page junctions. Memory locations will not be destroyed if the new start location is the same as the old start location.

A number of useful subroutines are also included as part of the programs. If you branch to location X'3CF8, the message "OK" will be printed, and if you branch to location X'3D0E, the message "FAULTY" will be printed. In both cases control will return to PIPBUG after the message is printed.

A message printing subroutine is included at locations X'3CCB to X'3CDC. This expects R1 and R2 to point to the start of a an ASCII message string. The string must be terminated by the null (X'00) character. If you enter this routine at location X'3CCB, the message will be printed on a new line, while if you enter at location X'3CCE, the message will be printed on the current line.

A subroutine called GPAR is located at address X'3C07. This uses the PIPBUG subroutine GNUM to get three parameters from the PIPBUG line buffer, and store them as bytes in locations X'2FFA to X'2FFF inclusive. The first parameter is stored in locations X'2FFA and X'2FFB, and is called START.

The second parameter is incremented, and then stored in locations X'2FFC and X'2FFD. It is called END. The third parameter, called NEW, is stored in locations X'2FFE and X'2FFF.

The subroutine INCRT is called at location X'3C2A, and increments the value START. It then compares START with END, and sets the condition code bits accordingly before returning. The condition code is set to "less than" (10) if START is less than END.

Another useful subroutine is called PADR, and is called at location X'3C3C. It will print the value of START, as a four digit hexadecimal number, at the start of a new line. The address is followed by a single space.

A number of smaller subroutines are also contained among the programs, but these are rather specialised, and will not be used very often. Interested readers can use the disassembler to disassemble the listing, and hence locate them.

To burn the program into a 2708, simply load it into a convenient area of RAM, and use the program supplied with the Prom Programmer article (Jan 1979) to copy it into the PROM. The program contains absolute addresses, and will only run at the correct locations, starting at X'3C00. RAM must exist at locations X'2FFA to X'2FFF inclusive.

Note that the listing of the programs given in this article shows them stored temporarily in the RAM at locations X'0600—07BD. This should be a convenient place to store them initially in most systems, before burning them into your PROM.